REMARKS/ARGUMENTS

Claims 1, 7, 15 and 19 have been amended to clarify the subject matter regarded as the invention. In particular, claim 7 has been clarified to recite a method reference portion for selected information, wherein the method reference portion includes one or more references cells which provide information associated with one or more methods which have been selected to be loaded into the virtual machine (see, for example, Specification, page 9, lines 20). It is respectfully requested that claim 2 be cancelled without prejudice or disclaimer. Claims 1, 3-21 remain pending.

In the Office Action, the Examiner objected to Fig 1. A proposed copy of Fig. 1 which has been labeled as "Prior Art," and includes a label 102, is hereby submitted herein for the Examiner's approval.

In addition, the Examiner has rejected claims under 35 U.S.C. §112 , 102 and 103. These rejections are fully traversed below.

Rejection of claims under 35 U.S.C. §112

In the office Action, the Examiner has rejected claims 3, 9 and 20 under 35 U.S.C. §112, second paragraph, as being indefinite for failing to particularly point out and distinctively claim the subject matter regarded as the invention. It is respectfully submitted that one skilled in the art can use a variety of different criteria to select information from a class file that is "likely to be used" by a virtual machine. By way of example, information that is likely to be used over a pre-determined threshold (e.g., at least 10% chance) can be used. One skilled in the art would also appreciate that the pre-determined threshold can be based on several other factors including system requirement (e.g., speed, memory). Moreover, the likelihood of use can be determined based on many different factors (e.g., type of information, past history, current use, etc.). In general, any selection that is made based on the likelihood of use can be interpreted as the scope of the claimed invention. In addition, one skilled in the art would know that populating a component (e.g., a method name field) generally operates to provide (e.g., fill into a blank space) the appropriate information for the component (e.g., the method name). In fact, the term "appropriate information or references to appropriate information," further clarifies the population operation in the claimed

invention. Accordingly, it is respectfully requested that the Examiner withdraw all rejections under 35 U.S.C. §112.

Rejection of claims under 35 U.S.C. §102

In the Office Action, the Examiner rejected claims 1-12, 17 and 21 under 35 U.S.C. §102 as being anticipated by U.S. Patent Application No. 6,339,841 (*Merrick et al.*).

The present application relates to techniques for loading class files into virtual machines. In accordance with one aspect of the invention, a mechanism for selectively loading information into a virtual machine can be provided. In one embodiment, a class file is initially loaded in its entirety into a memory portion (e.g., heap memory). However, only a selected portion of the class file is loaded into the virtual machine. As will be appreciated, this allows for a better use of resources available to a virtual machine, especially those that operate with limited memory resources (e.g., embedded systems).

As a representative claim, claim 1 pertains to a method of loading a class file into a virtual machine that operates in an object-oriented computing system. The class file is associated with a class. Initially, a class file is loaded into a memory portion of the computing system. It should be noted that the class file is loaded in its entirety without processing it. After the class file has been loaded into the memory portion, information from the class file is selected for loading into the virtual machine. Finally, the selected information is loaded directly form the memory portion. It should also be noted that information that has not been selected is not loaded into the virtual machine (claim 1).

It is noted that *Merrick et al.* relates to a class loading model. However, *Merrick et al.* teaches performing a post compilation process on object code to generate component object code, namely, Meta data and methods. The post compilation process is applied in order to break down the self-contained structure of a classes and convert a self contained class into individually accessible components of the class (*Merrick et al.*, Col. 3, lines 36-45). As such, it is respectfully submitted that *Merrick et al.* does not teach selecting information directly from the class file. Instead, *Merrick et al.* teaches: first breaking down the object code into individually accessible components

and then loading these individual components into the virtual machine (*Merrick et al*, Col. 3, lines 36-43).

Furthermore, it is respectfully submitted that there is no suggestion in *Merrick et al.* with respect to selecting and loading information directly from the class file. In fact, *Merrick et al.* teaches away from these features because the methodology of *Merrick* requires post compilation processing to generate Meta data and methods. In contrast, the claimed invention does not require any post compilation processing of the class file. Also, the methodology of *Merrick* does NOT teach initially loading into a memory portion of a client a class file in its entirety and then a client selecting components from the class file. Instead, post compilation is performed at a server and the client loads the components from the server. Hence, the methodology of *Merrick et al.* is substantially different than the claimed invention. Accordingly, it is respectfully submitted that *Merrick et al.* cannot possibly be combined with another reference to teach the claimed invention because its methodology is fundamentally different from the claimed invention. Thus, it is respectfully submitted that claim 1 is patentable over *Merrick et al.* for several reasons.

In addition, claims that are dependent on claim 1 are also patentable over *Merrick et al.* for at least these reasons. Moreover, these claims recite features that render them patentable over *Merrick et al.* for additional reasons. For example, claim 7 additionally recites loading an internal representation of the selected information that includes a method reference portion for the selected information, wherein the method reference portion includes one or more references cells which provide information associated with one or more methods which have been selected to be loaded into the virtual machine. It is noted that Fig. 2 of *Merrick et al.* illustrates a method table 26. However, it is respectfully submitted that the method table 26 of *Merrick et al.* does not teach or suggest a method reference portion that includes one or more references cells which provide information associated with one or more methods which have been selected to be loaded into the virtual machine. In addition, *Merrick et al.* does not teach or suggest populating these cells (see, for example, claim 9).

Still further, it is respectfully submitted that independent claims 15 and 19 are also patentable over *Merrick et al.* for similar reasons as discussed above. Moreover, claim 15, among other things, additionally recites determining whether a class file exists in a dedicated heap memory portion and loading the class file in the dedicated heap

memory portion when it is determined that said class file does not exist. It is respectfully submitted that *Merrick et al.* does not teach these features. This is believed to be evident because *Merrick et al.* does not even teach loading a class file into heap memory when a request is encountered.

Summary

Based on the foregoing, it is submitted that claims 1, 3-21 are patentably distinct over the cited art of record. Additional limitations recited in the independent claims or the dependent claims are not further discussed because the limitations discussed above are sufficient to distinguish the claimed invention from the cited art. Accordingly, Applicant believes that all pending claims are allowable and respectfully requests a Notice of Allowance for this application from the Examiner.

Applicants hereby petition for an extension of time which may be required to maintain the pendency of this case, and any required fee for such extension or any further fee required in connection with the filing of this Amendment is to be charged to Deposit Account No. 500388 (Order No. SUN1P815). Should the Examiner believe that a telephone conference would expedite the prosecution of this application, the undersigned can be reached at the telephone number set out below.

Respectfully submitted,
BEYER WEAVER & THOMAS, LLP

R. Mahboubian
Reg. No. 44,890

P.O. Box 778
Berkeley, CA 94704-0778
(650) 961-8300

Atty. Docket No.: SUNP815/P5613

SUBSTITUTE SPECIFICATION

WITH REDLINING

Attorney Docket No. SUN1P815/P5613

PATENT APPLICATION

TECHNIQUES FOR LOADING CLASS FILES INTO VIRTUAL MACHINES

Inventors: 1. Stepan Sokolov
34832 Dorado Common
Fremont, CA 94555
Citizenship: Ukraine

2. David Wallman
777 S. Mathilda Ave., #266
Sunnyvale, CA 94087
Citizenship: Israel

Assignee: Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303

BEYER WEAVER & THOMAS, LLP
P.O. Box 778
Berkeley, CA 94704-0778
Telephone (650) 961-8300

TECHNIQUES FOR LOADING CLASS FILES INTO VIRTUAL MACHINES

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to U.S. Patent Application No. 09/703,361 (Atty. Dkt. No. SUN1P809/P5500), entitled "IMPROVED FRAMEWORKS FOR INVOKING METHODS IN VIRTUAL MACHINES," which is hereby incorporated herein by reference.

This application is related to U.S. Patent Application No. 09/703,356 (Atty. Dkt. No. SUN1P810/P5510), entitled "IMPROVED METHODS AND APPARATUS FOR NUMERIC CONSTANT VALUE INLINING IN VIRTUAL MACHINES," which is hereby incorporated herein by reference.

This application is related to U.S. Patent Application No. 09/703,449 (Atty. Dkt. No. SUN1P814/P5417), entitled "IMPROVED FRAMEWORKS FOR LOADING AND EXECUTION OF OBJECT-BASED PROGRAMS," which is hereby incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. The Field of the Invention

The present invention relates generally to object oriented programming environments. More specifically, the invention relates to improved frameworks for loading class files into virtual computing machines.

2. The Relevant Art

Recently, the Java™ programming environment has become quite popular. The Java™ programming language is an object-based, high level programming language that is designed to be portable enough to be executed on a wide range of computers ranging from small devices (e.g., pagers, cell phones and smart cards) up to supercomputers. Computer programs written in the Java™ programming language (and other languages) may be compiled into Java™ virtual machine instructions

(typically referred to as Java™ bytecodes) that are suitable for execution by a Java™ virtual machine implementation.

The Java™ virtual machine is commonly implemented in software by means of an interpreter for the Java™ virtual machine instruction set, but in general may be software, hardware, or both. A particular Java™ virtual machine implementation and corresponding support libraries together constitute a Java™ runtime environment.

Computer programs in the Java™ programming language are arranged in one or more classes or interfaces (referred to herein jointly as classes or class files). Such programs are generally platform independent (i.e., hardware and operating system). As such, these computer programs may be executed unmodified on any computer that is able to run an implementation of the Java™ runtime environment. A class written in the Java™ programming language is compiled to a particular binary format called the "class file format" that includes Java™ virtual machine instructions for the methods of a single class. In addition to the Java™ virtual machine instructions for the methods of a class, the class file format includes a significant amount of ancillary information that is associated with the class. The class file format (as well as the general operation of the Java™ virtual machine) is described in some detail in The Java Virtual Machine Specification by Tim Lindholm and Frank Yellin (ISBN 0-201-31006-6), which is incorporated herein by reference.

Generally, when a class file is loaded into the virtual machine, the virtual machine essentially makes a copy of the class file for its internal use. The virtual machine's internal copy is sometimes referred to as an "internal class representation." In conventional virtual machines, the internal class representation is typically almost an exact copy of the class file. This is true regardless of whether the loaded information is likely to be used or is not used. For example, an exact copy of common Java™ classes (e.g., class PrintWriter), are loaded into the virtual machine. These classes typically have a large size. Thus, a common class, for example, class PrintWriter, may take up as much as 40 KiloBytes (40 K) of memory. However, typically, 90% of the class is not used during the execution of a computer program. This, of course, results in a grossly inefficient use of memory resources. In some circumstances, particularly in embedded systems which have limited memory resources, this inefficient use of memory resources is a significant disadvantage.

To further elaborate, Fig. 1 depicts a representation of a class file 100 inside a virtual machine. The class file 100 includes Methods A-Z portions that correspond to methods associated with a class. In addition, the class data 102 represents class data for the class. As will be appreciated by those skilled in the art, during typical execution of a program, only a small number of methods may be needed, for example, only Methods A and B may be needed. Nevertheless, all the methods associated with a class file are conventionally loaded. Similarly, only a data portion 104 may have been needed, but conventionally, all of the class data 102 is loaded. Thus, conventional techniques result in grossly inefficient use of memory resources which is a significant disadvantage, especially when memory resources are limited.

In view of the foregoing, improved techniques for loading class files into virtual computing machines are needed.

SUMMARY OF THE INVENTION

To achieve the foregoing and other objects of the invention, improved techniques for loading class files into virtual computing machines are disclosed. One aspect of the present invention seeks to provide a mechanism that will generally improve the efficiency of virtual machines by selectively loading information into a virtual machine. In other words, unlike conventional techniques where the entire class file is substantially loaded into the virtual machine, the inventive techniques can operate to load only a portion of the class file. As will be appreciated, this allows a better use of the resources of the virtual machine. The inventive mechanisms are especially effective in virtual machines that operate with limited memory resources (e.g., embedded systems). In one embodiment, class files suitable for loading into a virtual machine are initially loaded into a memory portion (e.g., heap memory). Then, information that is needed to be loaded into the virtual machine is selected. Finally, only the selected information is loaded into the virtual machine.

The invention can be implemented in numerous ways, including as a method, an apparatus, a computer readable medium, and a database system. Several embodiments of the invention are discussed below.

As a method for loading a class file into a virtual machine, one embodiment of the invention includes the acts of: loading the class file into a memory portion of the computing system; selecting information from the class file to be loaded into the virtual machine; and loading the selected information from the memory portion into the virtual machine and not loading information not selected from the class file into the virtual machine.

As another method for loading a class file into a virtual machine, another embodiment of the invention include the act of: encountering a request to use at least one method of a class associated with a class file; determining whether the class file exists in a dedicated heap memory portion; loading the class file in the dedicated heap memory portion when the determining determines that the class file does not exist in the dedicated heap memory portion; selecting information associated with the at least one method of the class; determining whether an internal representation of the class file exists in the virtual machine; creating an internal representation of the class file in the

virtual machine when the determining determines that an internal representation of the class file does not exist in the virtual machine; and loading into the virtual machine the selected information associated with the at least one method of the class and not loading into the virtual machine information that was not selected.

As a computer readable media, including computer program code for loading a class file into a virtual machine, one embodiment of the invention includes: computer program code for loading the class file into a memory portion of the computing system; computer program code for selecting information from the class file to be loaded into the virtual machine; and computer program code for loading the selected information from the memory portion into the virtual machine and not loading information not selected from the class file into the virtual machine.

These and other aspects and advantages of the present invention will become more apparent when the detailed description below is read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Fig. 1 depicts a representation of a class file inside a virtual machine;

Fig. 2 illustrates an object-oriented programming environment in accordance with one embodiment of the invention;

Fig. 3 is a block diagram of the internal class representation in accordance with one embodiment of the invention;

Fig. 4 is a diagrammatic representation of a reference cell in accordance with one embodiment of the present invention; and

Fig. 5 illustrates an exemplary loading method for loading a class file in accordance with one embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

As described in the background section, the Java™ programming environment has enjoyed widespread success. Therefore, there are continuing efforts to extend the breadth of Java™ compatible devices and to improve the performance of such devices. One of the most significant factors influencing the performance of Java™ based programs on a particular platform is the performance of the underlying virtual machine. Accordingly, there have been extensive efforts by a number of entities to provide improved performance to Java™ compliant virtual machines. In order to be Java™ compliant, a virtual machine must be capable of working with Java™ classes which have a defined class file format. Although it is important that any Java™ virtual machine be capable of handling Java™ classes, the Java™ virtual machine specification does not dictate how such classes are represented internally within a particular Java™ virtual machine implementation.

The present invention pertains to improved frameworks for loading class files into virtual computing machines. One aspect of the present invention seeks to provide a mechanism that will generally improve the efficiency of virtual machines by selectively loading information into a virtual machine. In other words, unlike conventional techniques where the entire class file is substantially loaded into the virtual machine, the inventive techniques can operate to load only a portion of the class file. As will be appreciated, this allows for a better use of the resources. The inventive mechanisms are especially effective in virtual machines that operate with limited memory resources (e.g., embedded systems). In one embodiment, class files suitable for loading into a virtual machine are initially loaded into a memory portion (e.g., heap memory). Then, information that is needed to be loaded from the class file into the virtual machine is selected. Finally, only the selected information from the class file is loaded into the virtual machine.

Embodiments of the invention are discussed below with reference to Figs. 2 - 4. However, those skilled in the art will readily appreciate that the detailed description

given herein with respect to these figures is for explanatory purposes as the invention extends beyond these limited embodiments.

Fig. 2 is a representation of an object-oriented computing environment 200 in accordance with one embodiment of the invention. The object-oriented computing environment 200 includes a class file 202, a raw-class heap portion 204, and an internal class representation 206 of the class file 202. The internal class representation 206 illustrates the information that is loaded in a virtual machine operating in the object-oriented computing environment 200.

As shown in Fig. 2, the class file 202 can include associated Methods M_1-M_n, as well as a data portion D. The class file 202 can be resident, for example, on a computer readable medium (e.g., compact disk, floppy disk, etc.) Furthermore, as will be appreciated, the class file 202 may be located anywhere in a distributed computing environment. Accordingly, in a distributed computing environment, the class file 202 may be transferred over a computer network from a remote location and can then be loaded into the virtual machine.

The raw-class heap portion 204 represents a portion of the memory of the object oriented computing environment 200 that is used to initially load the class file 202. This memory can be, for example, a memory portion of the object-oriented computing environment 200 that is dedicated to loading class files (i.e., dedicated memory). In contrast, the internal class representation 206 illustrates the information that is actually loaded inside the virtual machine. In this example, only the Methods M_i and M_j of the Methods M_1-M_n have been loaded into the virtual machine. Similarly, only a portion Di of the class data D is loaded into the virtual machine. Furthermore, as will be appreciated by those skilled in the art, maintenance can be performed on raw-class heap portion 204. For example, class files can be removed from the raw-class heap portion 204 on a Least Recently Used (LRU) basis.

Fig. 3 is a block diagram of the internal class representation 206 in accordance with one embodiment of the invention. The internal class representation 206 can be, for example, implemented as a data structure embodied in a computer readable medium that is suitable for use by a virtual machine. As shown in Fig. 3, the internal class representation 206 includes a method information portion 210. The method

information portion 210 can be arranged to contain or reference information relating to one or more methods. These methods can be, for example, Java™ implemented methods. Furthermore, as will be appreciated, the method information portion 210 can be implemented in various ways, for example, as a table similar to a table of method information implemented in a standard Java™ class file.

In addition to the method information portion 210, the internal class representation 200 includes a method reference portion 220 associated with the methods contained within the internal class representation 206. The method reference portion 220 can be arranged to include any reference cell associated with the class. Again, as will be appreciated, the method reference portion 220 can be implemented in a wide variety of different ways depending on the needs of a particular system. By way of example, in one embodiment, the reference cells are linked together using a link-list construct. Each reference cell can include information that is useful in invoking a method.

It should be noted that reference cells are created selectively (e.g., when it is certain and/or when it is likely that a method is to be invoked). Accordingly, in accordance with one embodiment of the invention, if a method does not appear to be invoked, reference cells corresponding to that method are not created. As a result, processing time and memory space may further be improved.

As noted above, the method reference portion 220 of Fig. 3 can be arranged to include any reference cell associated with the class. Fig. 4 is a diagrammatic representation of a reference cell 400 in accordance with one embodiment of the present invention. In the illustrated embodiment, the reference cell 400 includes a method name field 402, a method signature field 404, and a code reference field 406. Typically, a value stored in one of the fields 402, 404, and 406 can be referenced (i.e., point to another value). As noted above, a class may have one or more methods associated with it. A class can have one or more corresponding reference cells, for example, the reference cell 300. The method name field 402 is arranged to identify the name of a method associated with the class. This is typically done by storing the actual method name in the method name field 402. However, again, this can also be accomplished by storing a reference or index to the method name.

The signature field 404 is arranged to contain or reference a signature associated with the method corresponding the reference cell. The nature of the signature is well known to those familiar with method invocation in Java™ virtual machines. Typically, in most conventional Java™ virtual machines, the signature is constructed into a form usable by the virtual machine using a series of calls to the constant pool at runtime when the method is invoked. Although this works well, it is relatively slow. One advantage to including the signature in the reference cell is that a signature suitable for direct use by the virtual machine can be constructed during loading and either stored directly in the reference cell or stored in a location that is referenced by the reference cell. In the described embodiment, the reference cell contains a reference (e.g., pointer or index) to the signature rather than actually storing the signature, simply because signatures can be relatively large and their relative sizes may vary significantly for different classes. In addition, references to signatures can be used across reference cells. Thus, referencing the signature tends to be a more efficient use of memory. More details about internal representation of methods can be found in related U.S. Patent Application No. 09/703,361 (Atty. Dkt. No. SUN1P809/P5500), entitled "IMPROVED FRAMEWORKS FOR INVOKING METHODS IN VIRTUAL MACHINES," U.S. Patent Application No. 09/703,356 (Atty. Dkt. No. SUN1P810/P5510), entitled "IMPROVED METHODS AND APPARATUS FOR NUMERIC CONSTANT VALUE INLINING IN VIRTUAL MACHINES," and U.S. Patent Application No. 09/703,449 (Atty. Dkt. No. SUN1P814/P5417), entitled "IMPROVED FRAMEWORKS FOR LOADING AND EXECUTION OF OBJECT-BASED PROGRAMS," all of which are hereby incorporated herein by reference.

Finally, the code reference field 406 is arranged for referencing the code associated with the method. As will be appreciated by those skilled in the virtual machine art, there is often code associated with a method that the virtual machine executes at runtime. The code reference field 406 simply provides a place to identify the location where such information can be or is stored. It should also be noted that the code reference field 406 can be loaded when it is determined that there is a need for the code. Similarly, method name field 402 and method signature field 404 can selectively be loaded.

Fig. 5 illustrates an exemplary loading method 500 for loading a class file in accordance with one embodiment of the invention. The method 500 can be implemented in an object oriented computing environment to selectively load class files into a virtual machine. For the sake of illustration, in the described embodiment, the loading method 500 is used in a Java™ runtime environment to load a Java™ class file. Initially, at operation 502, a request to use a class is encountered. Next, at operation 504, a determination is made as to whether an internal representation of the class exists in the virtual machine. If it is determined at operation 504 that an internal class representation does not exist for the class, the method 500 proceeds to operation 506 where the appropriate class file (i.e., class file associated with the class) is loaded into the raw-class heap. After the appropriate class file is loaded into the raw-class heap, at operation 508, an internal representation of the class is created. This internal representation can be, for example, the internal representation 206 illustrated in Fig. 3. Next, at operation 510, the required components of the class are identified and extracted from the raw-class heap. As will be appreciated, these components can be selected, for example, when the class is resolved. Accordingly, only the required components of the class need to be extracted after being identified. Thereafter, the extracted components of the class are loaded into the virtual machine at operation 512. This can include loading required methods and creating references for loaded methods in a method information portion (e.g., similar to a method table implemented in a standard Java™ class). The method 500 ends following operation 512.

On the other hand, if it is determined at operation 504 that an internal class representation exists for the class, the method 500 proceeds to operation 514 where a determination is made as to whether a requested method of the class already exists in the internal class representation in the virtual machine. If it is determined at operation 514 that the requested method of the class already exists in the internal class representation in the virtual machine, the method 500 ends. However, if it is determined at operation 514 that the requested method does not exist in the raw-class heap, the method 500 proceeds to operation 516 where it is determined whether the requested class exists in the raw-class heap. If it is determined at operation 514 that the requested method does not exist, the method 500 proceeds to operation 506 where the appropriate class file is loaded in the raw-class heap. However, if it is determined at operation 516 that the requested class exists in the raw-class heap, the method 500

proceeds directly to operation 510, bypassing operations 506 and 508. At operation 510, the required components of the class are identified and extracted from the raw-class heap. Thereafter, the extracted components of the class can be loaded into the virtual machine at operation 512. As noted above, this can include loading required methods and creating references for loaded methods in a method information portion (e.g., similar to a method table implemented in a standard Java™ class). The method 500 ends following operation 512.

The many features and advantages of the present invention are apparent from the written description, and thus, it is intended by the appended claims to cover all such features and advantages of the invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction and operation as illustrated and described. Hence, all suitable modifications and equivalents may be resorted to as falling within the scope of the invention.

What is claimed is:

CLAIMS

1. A method of loading a class file into a virtual machine, said class file being associated with a class, and said virtual machine operating in an object-oriented computing system, said method comprising:

loading said class file into a memory portion of the computing system;

selecting information from said class file to be loaded into said virtual machine; and

loading said selected information from said memory portion into said virtual machine and not loading information not selected from the class file into the virtual machine.

2. A method as recited in claim 1, wherein the method further comprises:

encountering a request to a said class associated with a class file.

3. A method as recited in claim 1, wherein said selecting of information operates to select information from said class file that is likely to be used by the virtual machine.

4. A method as recited in claim 1, wherein said selecting of information operates to select information from said class file that is needed to be used by the virtual machine.

5. A method as recited in claim 1, wherein said selecting of information operates to select information that includes information associated with at least one method of said class.

6. A method as recited in claim 1, wherein said loading of only said selected information operates to create an internal representation of the class file in the virtual machine.

7. A method as recited in claim 6, wherein said internal representation of said class file includes a method reference portion.

8. A method as recited in claim 7, wherein said method reference portion includes a method name field, a method signature field, and a method code field.

9. A method as recited in claim 1, wherein said loading of only said selected information operates to populate said method name field, method signature field, and method code field with appropriate information or references to appropriate information.

10. A method as recited in claim 1, wherein said memory is a heap memory of said computing system.

11. A method as recited in claim 1, wherein the method further comprises:

determining whether an internal representation of the class file exists in the virtual machine; and

creating an internal representation of the class file in the virtual machine when said determining determines that an internal representation of the class file does not exist in the virtual machine.

12. A method as recited in claim 1, wherein the method further comprises:

determining whether said class file exists in said memory portion; and

loading said class file in said memory portion when said determining determines that said class file does not exist in said memory portion.

13. A method as recited in claim 1, wherein the method further comprises:

removing said class file from said memory portion.

14. A method as recited in claim 13, wherein said removing is performed on a Least Recently Used basis.

15. A method of loading a class file into a virtual machine, said class file being associated with a class, and said virtual machine operating in a computing system, said method comprising:

encountering a request to use at least one method of a class associated with a class file;

determining whether said class file exists in a dedicated heap memory portion;

loading said class file in said dedicated heap memory portion when said determining determines that said class file does not exist in said dedicated heap memory portion;

selecting information associated with said at least one method of said class;

determining whether an internal representation of the class file exists in said virtual machine;

creating an internal representation of the class file in the virtual machine when said determining determines that an internal representation of the class file does not exist in said virtual machine; and

loading into the virtual machine said selected information associated with said at least one method of said class and not loading into said virtual machine information that was not selected.

16. A method as recited in claim 15, wherein the method further comprises:

removing said class file from said dedicated heap memory portion on a Least Recently Used basis.

17. A method as recited in claim 15, wherein said selected information includes a method name field, a method signature field, and a method code field with appropriate information or references to appropriate information.

18. A method as recited in claim 15, wherein said internal representation includes a reference cell associated with said at least one method.

19. A computer readable media, including computer program code for loading a class file into a virtual machine, said class file being associated with a class, and said virtual

machine operating in an object-oriented computing system, said computer program code comprising:

computer program code for loading said class file into a memory portion of the computing system;

computer program code for selecting information from said class file to be loaded into said virtual machine; and

computer program code for loading said selected information from said memory portion into said virtual machine and not loading information not selected from the class file into the virtual machine.

20. A computer readable media as recited in claim 19, wherein said computer program code for selecting of information operates to select information from said class file that is likely to be used by the virtual machine.

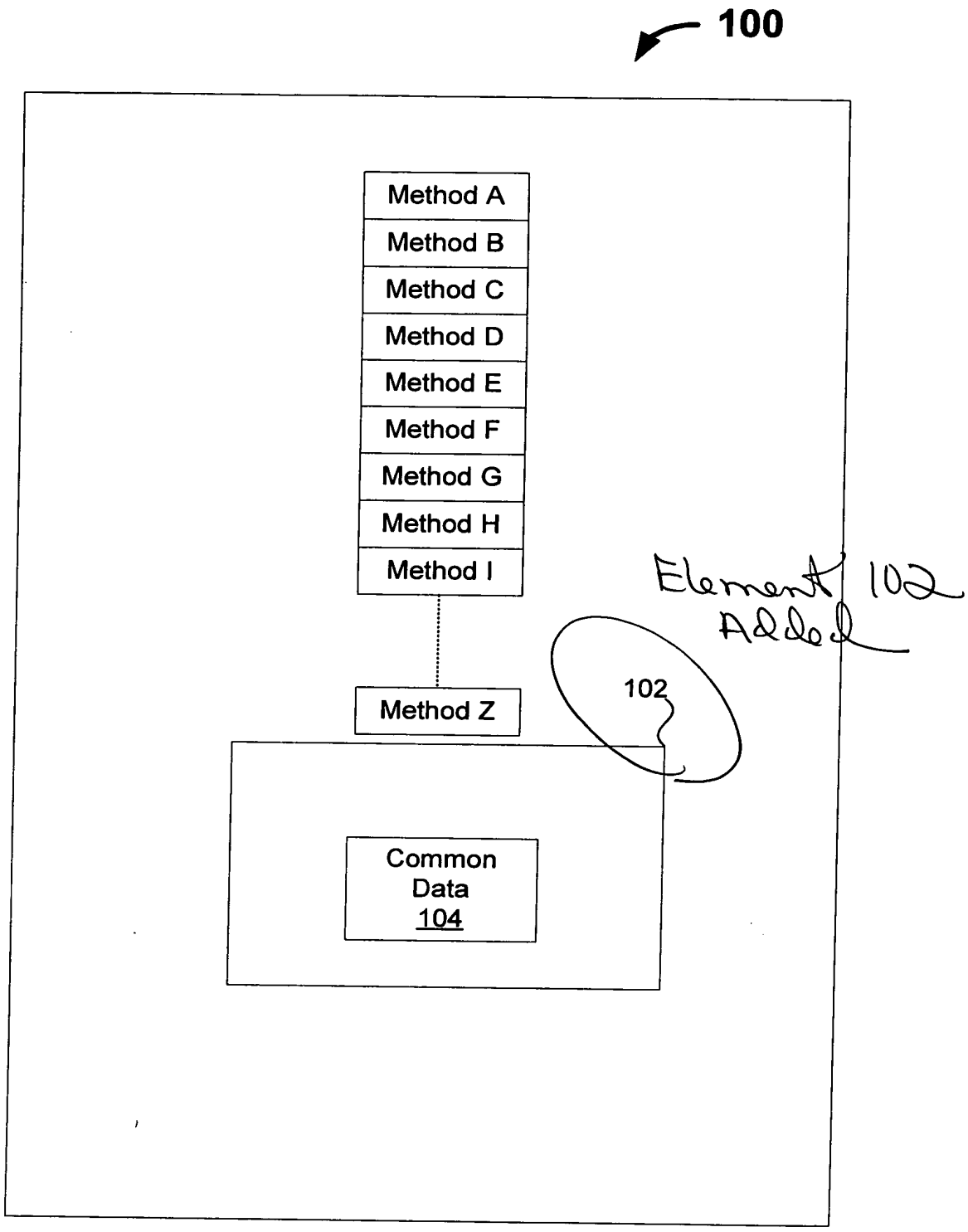
21. A computer readable media as recited in claim 19, wherein said computer program code for selecting of information operates to select information from said class file that is needed to be used by the virtual machine.

TECHNIQUES FOR LOADING CLASS FILES INTO VIRTUAL MACHINES

ABSTRACT OF THE DISCLOSURE

Improved techniques for loading class files into virtual computing machines are disclosed. These techniques provide a mechanism that will generally improve the efficiency of virtual machines by selectively loading information into a virtual machine. As will be appreciated, this allows a better use of the resources of the virtual machine. This is especially effective in virtual machines that operate with limited memory resources (e.g., embedded systems). In one embodiment, class files suitable for loading into a virtual machine are initially loaded into a memory portion (e.g., heap memory). Then, information that is needed to be loaded into the virtual machine is selected. Finally, only the selected information is loaded into the virtual machine.

ANNOTATED SHEET SHOWING CHANGES



(Prior Art)

Fig. 1